

#### REMARKS

Claims 27-29, 33, 36, and 43-44 stand rejected under 35 U.S.C. 102(e) as being anticipated by Krishnan (U.S. Pat. No. 6,141,698). Claims 30-32, 37-40, and 45-46 stand rejected under 35 U.S.C. 103(a) as being unpatentable over Krishnan in further view of Bell et al. (U.S. Pat. No. 5,727,205).

The present invention, as defined by amended claim 27, relates to a computer readable medium having an executable application recorded thereon. The executable application comprises a program, one or more encrypted sub-routines, and a decryption routine. The program is executed in response to execution of the executable application by a computer system. The program requires access to the sub-routines during execution. The decryption routine operates to detect whether a required sub-routine is already available within the computer system, to cause the program to use the sub-routine within the computer system if already available, and, at least when access to the sub-routine is required by the program, to decrypt the required encrypted sub-routine into an executable form if the sub-routine is not already available within the computer system.

Krishnan describes a method and system for injecting new code into an already existing executable file (e.g. col. 2, lines 31-33 of Krishnan). The new code that is inserted consists of a reference to a dynamic link library (DLL). Accordingly, when the executable file is executed, the injected reference causes the DLL to be loaded. Importantly, the injected code consists of a reference to a DLL, not the DLL itself (e.g. col. 3, lines 60-66 of Krishnan). A DLL, as that term is commonly understood, is a shared file that is intended to be used by more than one software application. For example, a DLL responsible for providing a 'Print Preview' of a document, may be used by both a word processing application and also a web browser application. It is therefore important that the DLL remain available to other software applications and it is for this reason that the injected

code consists of a reference to the DLL file rather than the DLL file itself.

There are two methods by which Krishnan's reference to the DLL may be injected. In the first, the import table of the executable file is modified so as to contain reference to the DLL. Consequently, during execution of the executable file, the DLL file is imported into memory. In the second, 'loader code' is injected into the executable file such that the DLL is loaded into memory during execution (col. 5, lines 59-63 and col. 6, lines 49-54). Nevertheless, in both embodiments, the DLL does not form part of the executable file. This can be clearly seen in, for example, FIG. 1 of Krishnan in which the DLL (labelled 107) is distinct from the executable file (labelled 106).

In addition to injecting a reference to a DLL, Krishnan also describes the injection of 'security code' within the executable file. The intention of the security code is to prevent modification of the executable file in a manner which circumvents the injected reference to the DLL. The security code may perform checksum checks on portions of the executable file and the DLL. Additionally, the security code may decrypt previously encrypted portions of the executable file (e.g. col. 2, lines 42-53 of Krishnan). Importantly, only portions of the executable file are encrypted; there is no suggestion of encrypting portions of the DLL. Indeed, to encrypt portions of the DLL would prevent the DLL from being used by other applications.

When injecting the security code, Krishnan's encryption routine encrypts portions of the executable file (e.g. col. 11, lines 32-37). The portions of the executable file that are encrypted are carefully chosen according to criteria set out between col. 13, line 18 and col. 15, line 2 of Krishnan. Although there is no suggestion of encrypting portions of the executable file that correspond to subroutines, it is conceivable that a subroutine might nevertheless be encrypted (owing to the relatively broad scope that the term 'subroutine' carries).

The amendments to claim 27 incorporate the subject-matter of previously presented claim 30 (now cancelled). Consequently, claim 27 now indicates that the decryption routine determines whether required subroutines are already available on the computer system and decrypts an encrypted subroutine in the event that a subroutine is not already available on the computer system.

The examiner concedes that claim 30 is not anticipated by Krishnan, but argues that the claim is rendered obvious by Krishnan in view of Bell (US 5,727,205).

As indicated by the examiner, Bell describes an application for installing software from a floppy disk to a computer system. The application determines whether files to be installed are already present on a computer system. If a file to be installed is already present, the application checks the version date of the file on the system and compares it against the version date of the file on the floppy disk. The application then replaces the file on the system with that on the floppy disk in the event that the system file is older than the floppy disk file.

The examiner argues that it would be obvious to a person of ordinary skill in the art to modify the system of Krishnan such that the injected security code checks for local copies of encrypted subroutines and to decrypt an encrypted subroutine only if there is no local copy. The examiner indicates that a person of ordinary skill in the art would be motivated by the need to prevent unnecessary processing that arises from decrypting of a subroutine that is already present on the system. Applicant respectfully disagrees.

The intention of the security code of Krishnan is to prevent modification of the executable file in a manner which circumvents the injected reference to the DLL. In particular, the security code is intended to prevent a user from obtaining an original copy of the executable file before the reference to the DLL and the security code were injected (e.g. col. 2, lines 50-56 of

Krishnan). To this end, the decryption routine of the security code decrypts only one encrypted portion of the file at any one time. In this way, a fully decrypted copy of the executable file is never available (e.g. col. 3, lines 4-8 of Krishnan). If unencrypted portions of the executable file were stored on the system, as is proposed by the examiner, a user could easily obtain an original copy of the executable file. Modifying the security code such that the decryption routine checks for unencrypted portions outside of the executable file would therefore defeat the very purpose of the security code. It would not therefore be obvious to the skilled person to modify the security code in the manner proposed by the examiner since to do so would contradict the very teachings of Krishnan and render the invention useless.

The examiner suggests that the skilled person would be motivated by the need to prevent unnecessary processing that arises from decrypting a subroutine that is already present on the system. However, Krishnan makes clear that only small portions of the executable file are encrypted such that the processing involved in decryption is small and that the speed of decryption is quick. Moreover, including the sequence of checking for unencrypted portions and then decrypting only in the event that an unencrypted portion is not available, as proposed by the examiner, is likely to consume more processing resources owing to the fact that the encrypted portions are small. It is therefore further submitted that the skilled person would not be motivated in the manner proposed by the examiner.

Consequently, (1) it would not be obvious to modify the security code of Krishnan as proposed by the examiner since to do so would contradict the very purpose of the security code and the very teachings of Krishnan, and (2) a person of ordinary skill in the art would not be motivated to modify the security code of Krishnan in the manner proposed by the examiner.

Applicant submits that, for the reasons provided above, the

invention defined by amended claim 27 is neither disclosed nor suggested by Krishnan and Bell, whether taken singly or in combination. Applicant therefore submits that amended claim 27 of the present application is patentable. It follows that dependent claims 28, 29, and 31-36 are also patentable.

The arguments provided above are equally applicable to independent claim 37 and independent claim 43, which has been amended to incorporate the subject-matter of claim 45 (now canceled). Applicant therefore submits that the subject matter defined by claims 37 and 43 is neither disclosed nor suggested by Krishnan and Bell, whether taken singly or in combination. Applicant therefore submits that amended claims 37 and 43 of the present application are patentable. It follows that dependent claims 38-42, 44, and 46-48 are also patentable.

Claim 31 has been revised so as to depend upon claim 27, and a minor revision has been made to claim 28. Additionally, claims 32, 40 and 46 have been amended to improve their clarity.

New claims 49 and 50 have been added. These claims correspond substantially to original claims 27 and 29. However, claim 49 includes the additional feature that the subroutines are shared subroutines (e.g. DLLs) that may be accessed by a further program. Basis for this additional feature may be found at, for example, page 5, final 6 lines.

As has been noted above, the executable file of Krishnan does not include a copy of the DLL but instead includes a reference to the DLL. Upon executing the executable file, the injected reference causes the shared DLL to be loaded into memory. Krishnan does not therefore describe an application that includes a shared subroutine. Moreover, there is no suggestion in Krishnan of encrypting the DLL. Instead, only small portions of the executable file (i.e. the program) are encrypted. Consequently, Krishnan additionally fails to describe a shared subroutine that is encrypted.

None of the other prior art documents cited by the examiner

teach or suggest an application that includes a program, encrypted subroutines and a decryption routine, wherein the subroutines are shared subroutines that may be accessed by other programs when decrypted. Applicant therefore submits that claim 49 is not anticipated by the prior art and in particular Krishnan.

Referring again to Krishnan, a DLL is a shared file that is intended to be used with more than one software application. It would not therefore be obvious to include the DLL within the executable file of Krishnan since the DLL could not then be available to other applications. Additionally, it would not be obvious to include a copy of the DLL with the executable file of Krishnan since there would then be two copies of the DLL (one copy within the executable file and the other copy stored separately on the system), which would consume storage space. Indeed, the very purpose of shared files is that only one copy need be provided, thereby saving storage space. It is for these very reasons that Krishnan refers to injecting a reference to a DLL. It is therefore submitted that it would not be obvious to a person of ordinary skill in the art to include the DLL within the executable file of Krishnan.

Even if a person of ordinary skill in the art were to include the DLL within the executable file, Krishnan clearly teaches that only small portions of the executable file are encrypted (e.g. col. 11, lines 34-39 of Krishnan). The ordinarily skilled person would not therefore contemplate encrypting all of the DLL since this would contradict the very teachings of Krishnan. In particular, decryption would be much slower and the user would experience a delay during execution of the executable file. Accordingly, even if a person of ordinary skill in the art were to include the DLL in the executable file, he would understand from Krishnan that only small portions of the DLL, if any, should be encrypted. It is therefore submitted that it would not be obvious to encrypt the entire DLL.

Applicant submits that the invention defined by claim 49 is neither disclosed nor suggested by Krishnan, when considered alone or in combination with any other prior art document. Applicant therefore submits that new claim 49 is patentable.

With regard to claim 50, the examiner argues at the top of page 4 of the office action that the features of this claim are disclosed by Krishnan at col. 12 line 48 to col. 13, line 4. This passage of Krishnan concerns encrypting portions of the executable file. Krishnan states that the encryption algorithm is preferably chosen such that the size of the encrypted data corresponds exactly to the original data. The original data is then replaced by the encrypted data (col. 13, lines 9-14 of Krishnan). Because the original and encrypted data are identical in size, relative addresses within the executable file are unchanged. For example, if the executable file included an instruction to jump to a particular address located after the encrypted data, the instruction would continue to execute correctly because the encrypted and original data correspond in size. However, Krishnan also states that the encryption algorithm may be chosen such that the size of the encrypted data differs from the original data (col. 12, lines 64-67). In this case, when the encrypted data is injected into the executable file, the locations of subsequent instructions within the executable file will change. Accordingly, the instruction to jump to an address after the encrypted data will not execute correctly unless the address is modified. Accordingly, as indicated by Krishnan, the differences in size between encrypted and original data must be tracked and used to adjust relocatable addresses.

However, there is nothing in Krishnan to suggest that the decryption routine decrypts a subroutine and then makes an entry in an address table such that the subroutine is readily available to the program (and other programs) whenever access to the subroutine is required. Indeed, Krishnan clearly indicates that the decryption routine decrypts only one portion of data at a

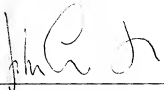
time and that the decrypted data is immediately overwritten by the next decrypted portion of data (e.g. col. 3, lines 4-8 of Krishnan). It is therefore clear that the decryption routine does not store a decrypted subroutine or make an entry in an address table such that the subroutine can be subsequently accessed whenever required.

Applicant therefore submits that claim 50 is not anticipated by Krishnan independently of the reasons discussed above regarding claim 49.

It would not be obvious to a person of ordinary skill in the art to modify the decryption routine of Krishnan since the security code would not longer function as intended. In particular, were the decryption routine modified so as to store a decrypted subroutine and make an entry in an address table such that the subroutine can be subsequently accessed whenever required, it would be relatively straightforward for a user to obtain a full unencrypted copy of the executable file by inspecting the address table and retrieving the decrypted subroutines.

Applicant therefore submits that the present invention as defined by claim 50 is neither disclosed nor suggested by Krishnan or any other prior art document, whether taken singly or in combination.

Respectfully submitted,

A handwritten signature in dark ink, appearing to read "John Smith-Hill", is written over a horizontal line.

John Smith-Hill  
Reg. No. 27,730

SMITH-HILL & BEDELL, P.C.  
16100 N.W. Cornell Road, Suite 220  
Beaverton, Oregon 97006

Tel. (503) 574-3100  
Fax (503) 574-3197  
Docket: FORR 2275